# List Scheduling: The Price of Distribution

Marc Tchiboukdjian, Denis Trystram, Jean-Louis Roch, Julien Bernard

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *List Scheduling: The Price of Distribution*

Marc Tchiboukdjian — Denis Trystram — Jean-Louis Roch — Julien Bernard

**N° 7208**

February 2010

Distributed and High Performance Computing


ISSN 0249-6399 ISRN INRIA/RR--7208--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

# List Scheduling: The Price of Distribution

Marc Tchiboukdjian* , Denis Trystram† , Jean-Louis Roch‡ , Julien Bernard§

Theme : Distributed and High Performance Computing
Équipe-Projet MOAIS

Rapport de recherche n° 7208 — February 2010 — 20 pages

**Abstract:** Classical list scheduling is a very popular and efficient technique for scheduling jobs in parallel and distributed platforms. It is inherently centralized. However, with the increasing number of processors in new parallel platforms, the cost for managing a single centralized list becomes too prohibitive. A suitable approach to reduce the contention is to distribute the list among the computational units. Thus each processor has only a local view of the work to execute.
The objective of this work is to study the extra cost that must be paid when the list is distributed among the computational units. We present a general methodology for computing the expected makespan based on the analysis of an adequate potential function which represents the load unbalance between the local lists. It is applied to several scheduling problems, namely, for arbitrary divisible load, for unit independent tasks, for weighted independent tasks and for tasks with dependencies. It is presented in detail for the simplest case of divisible load, and then extended to the other cases. More precisely, we prove that the time for scheduling a global workload $W$ on $m$ processors is equal to $W/m$ with an additional term in $\frac{4e}{e-1}\log_2 W$. We provide a lower bound which shows that this is optimal up to a constant factor in $\log_2 W$. This result is extended to the case of independent unit tasks and independent weighted tasks using a similar analysis. Moreover, we show how to adapt this methodology to the case of precedence task graphs. This analysis allows to improve the constant factor in the additive term of the bound of Arora, Blumofe and Plaxton. We finally provide some experiments using a simulator. We are able to fit the distribution of the makespan with existing probability laws. Moreover, we study the additive factor and show that it is around $3\log_2 W$ which indicate that our analysis is tight up to a factor less than 3.

**Key-words:** Scheduling, List algorithms, Distributed data, Work stealing

* marc.tchiboukdjian@imag.fr, CNRS and CEA/DAM,DIF
† denis.trystram@imag.fr, Grenoble University
‡ jean-louis.roch@imag.fr, Grenoble University
§ julien.bernard@lifc.univ-fcomte.fr, Université de Franche-Comté

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

# Ordonnancement par liste : cout de la répartition de la liste

**Résumé :** Les algorithmes de liste sont une technique classique et efficace pour l'ordonnancement de tâches sur des systèmes parallèles et distribués. Cette approche est fondamentalement centralisée. A cause de l'augmentation du nombre de processeurs dans les nouvelles plateformes parallèles, le coût de gestion d'une seule liste centralisée devient prohibitif. Une technique pour diminuer la contention est de répartir la liste sur les unités de calculs. Dans ce cas, chaque processeur a seulement une vue locale du travail à exécuter.
L'objectif de cet article est d'étudier le surcout induit par la répartition de la liste de tâches sur les différents processeurs. On présente une méthode générale pour calculer l'espérance du temps de complétion basée sur l'analyse d'une fonction potentielle qui représente le déséquilibre de charge entre les listes locales. On applique cette méthode à plusieurs problèmes d'ordonnancement : travail divisible, tâches unitaires indépendantes, tâches pondérées indépendantes et tâches avec dépendances. On présente dans un premier temps la méthode dans le cas simple du travail divisible puis on l'étend aux autres cas. On prouve que le temps de calcul d'une charge globale $W$ sur $m$ processeurs est égal à $W/m$ plus un terme additionnel en $\frac{4e}{e-1}\log_2 W$. On présente une borne inférieure qui montre que ce résultat est optimal à un facteur constant près en $\log_2 W$. On étend ce résultat au cas des tâches indépendantes avec une analyse similaire. De plus, on montre comment adapter cette méthode aux cas des graphes de dépendances. Cette analyse permet d'améliorer le facteur constant dans le terme additif de la borne de Arora, Blumofe et Plaxton. Enfin, on présente des résultats expérimentaux en utilisant un simulateur. La distribution du temps de complétion correspond à une loi de probabilité connue. De plus, on étudie le terme additif et on montre qu'il est environ de $3\log_2 W$ indiquant que notre analyse est précise à moins d'un facteur 3.

**Mots-clés :** Ordonnancement, Algorithmes de liste, Vol de Travail, Données distribuées

# 1 Introduction

Scheduling is a key issue while designing efficient parallel algorithms. The problem is to distribute the tasks of an application (load) among available computational units and determine at what time they will be executed. In most cases, the target objective is to minimize the completion time of the latest task to be executed (called the *makespan* and usually denoted by $C_{\max}$). It is a hard challenging problem which received a lot of attention during the last decades [17]. Two new books have been published few months ago on the topic [11, 20], which shows how still active the area is.

List scheduling is one of the most popular technique for scheduling the tasks of a parallel program. This algorithm has been introduced by Graham [14] and was used with profit in many further works (for instance in ETF [15] with communication times, in HEFT [23] for heterogeneous (unrelated) processors, for uniform machines in [10], for parallel rigid jobs in [22]). Its principle is to build a list of ready tasks and schedule them as soon as there exist available resources. Lists scheduling are low-cost (greedy) algorithms that are not too far from optimal solutions. Most proposed list algorithms differ in the way of considering the priority of the tasks for building the list, but they always consider a centralized management of the list. However, today the parallel and distributed platforms involve more and more processors. Thus, the time needed for managing such a centralized data structure can not be ignored anymore.

We describe below the underlying computational model and define informally the problem to solve. The parallel system is composed of a set of $m$ interconnected identical processors, that must execute a set of tasks. The total amount of load is $W$ and can represent a global divisible load, $W$ unit independent tasks, weighted tasks whose sum is equal to $W$, tasks with dependencies. Each processor owns its local queue of ready tasks and no one has a global view of the system. When the processor becomes idle, it can make a request of load to other processors in order to get some tasks. If the request fails, another request can be sent at the next time slot. Otherwise, a certain amount of load is transfered to this processor. There is no explicit communication cost in this model, it is implicitly taken into account in the cost of a load request. A load transfer is done in constant time, independently of the size of the load. This assumption is not restrictive: for the case of independent tasks, the description of a large number of tasks can be very short. For instance a whole subpart of an array of tasks can be represented in a compact way by the range of the corresponding indices, each cell containing the effective description of a task (a STL transform in [24]). For more general cases with dependencies, it is usually enough to transfer a root task which represents a part of the graph [3]. More details are provided in section 3.

In other words, the problem can be considered as a distributed version of the classical problems $P||C_{\max}$ and $P|prec, p_j = 1|C_{\max}$ [17].

**Related works.** Most related works dealing with scheduling consider centralized list algorithms. However, at execution time, the cost for managing the list is neglected. Practically, implementing such schedulers induces synchronization overheads when several processors access the list concurrently. To our knowledge, the only approach that takes into account this extra management cost is *work stealing* [9] (denoted by WS in short).

Contrary to classical centralized scheduling techniques, WS is a distributed algorithm. Each processor manages its own list of tasks. When a processor becomes idle, it randomly chooses another processor and requests some work. WS has been imple-

mented in many languages and parallel libraries including Cilk [12], TBB [21] and KAAPI [13]. It has been analyzed in a seminal paper of Blumofe and Leiserson [9] where they show that the expected makespan of series-parallel precedence graph with unit tasks is bounded by $\mathbb{E}[C_{\max}] \leq W/m + \mathrm{O}(T_\infty)$ where $T_\infty$ is the critical path of the graph. However the precedence graph is constrained to have only one source and out-degree at most 2 which does not easily model the basic case of independent tasks. Simulating independent tasks with a binary tree of dependencies gives a bound of $W/m + \mathrm{O}(\log W)$ as a complete binary tree of $W$ nodes has a depth of $T_\infty \leq \log_2 W$. However, with this approach, the structure of the binary tree dictates which tasks are stolen. Our approach achieves a similar bound with a better constant and processors are free to choose which tasks to steal. This is essential when tasks have affinity to certain processors because of cache considerations for instance [1]. Notice that there exists other ways to analyze work stealing where the work generation is probabilist and that targets steady state results [5, 18].

Another related approach which deals with load balancing is *balls into bins* games [4, 8]. The principle is to study the maximum load when $n$ balls are randomly thrown into $m$ bins. This is a simple distributed algorithm which is far from the scheduling problems we are interested in. First, it seems hard to extend this kind of analysis for tasks with dependencies. Second, as the load balancing is done in one phase at the beginning, the cost of computing the schedule is not considered. Some works [2] study parallel allocations but still do not take into account contention on the bins. Our approach, like WS, considers contention on the queues. Processors that request load on the same remote queue are in competition and only one can succeed.

Some works have been proposed for the analysis of algorithms in data structures and combinatorial optimization (including variants of scheduling) using potential functions. Our analysis is also based on a potential function representing the load unbalance between the local queues. This technique has been successfully used for analyzing basic load balancing [6] and WS [3] which improved the result of [9].

There exist several other connected works for instance convergence time to Nash equilibria in game theory [6] or load diffusion on graphs [7] that are not presented in this paper.

**Contributions.** List scheduling is centralized in nature. The purpose of this work is to study the effects of decentralization on list scheduling. The main result is to present a new framework for analyzing the distributed list scheduling algorithm (DLS). Based on the analysis of the load balancing between two processors during a work request, the total expected number of work requests can be deduced and a bound on the expected makespan is derived.

The methodology is generic and it is applied in this paper on several relevant variants of the scheduling problem. We show first that the expected makespan of DLS applied on $W$ unit independent tasks is equal to the absolute lower bound $W/m$ plus an additive term in $\frac{4e}{e-1}\log_2 W \leq 6.33 \log_2 W$. We propose a lower bound which shows that the analysis is tight up to a constant factor. We take into account the fact that tasks are not fully divisible which makes the analysis difficult when the number of tasks per queue is small. Second, we extend the previous analysis to the weighted case with unknown processing times. The additive term becomes $\mathrm{O}(\frac{p_{\max}}{p_{\min}} \cdot \log W)$ where $p_{\min}$ and $p_{\max}$ are the extremal values of the processing times. Third, we provide a new analysis for the WS model for scheduling DAGs from [9] that slightly improves the bound on the number of work requests to $\frac{4e(m-1)}{e-1} T_\infty \leq 6.33 m T_\infty$. Fourth, we

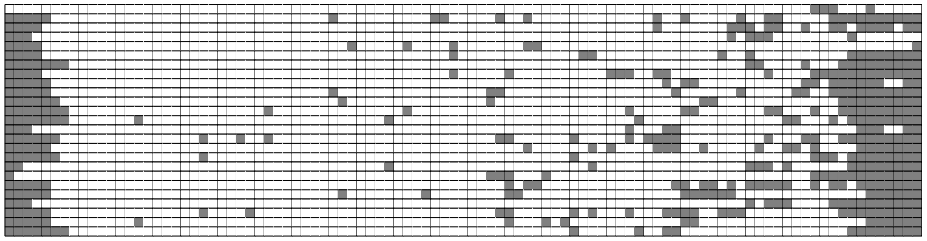


Figure 1: A typical execution of $W = 2000$ unit independent tasks on $m = 25$ processors using distributed list scheduling. Grey area represents idle times.

show that a balanced initial allocation induces less work requests. Finally, we give statistical evidence that the makespan of DLS on independent tasks follows a known distribution. Moreover, the experiments show that the theoretical analysis for independent tasks is almost tight: the overhead to $W/m$ is less than a factor of 3 away of the exact value.

**Roadmap.** We start by presenting the principle of the analysis in section 4, we apply this analysis on the simplest model, namely the *divisible load* model in section 5 and we adapt the proof for unit independent tasks in section 6. Then we extend the analysis for weighted independent tasks in section 7 and for tasks with dependencies in section 8. Section 9 discusses the initial repartition of tasks. We present and analyze simulation experiments in section 10.

# 2 Classical List Scheduling

Let us recall briefly the principle of list scheduling as it was introduced by Graham [14]. The problem is to schedule a precedence graph of $n$ tasks whose processing times are denoted $p_j$ for $1 \leq j \leq n$. The analysis states that the makespan of any list algorithm is not greater than twice the optimal makespan. One way of proving this bound is to use a geometric argument on the Gantt chart: $mC_{\max} = \sum_{1 \leq j \leq n} p_j + S_{\text{idle}}$ where the last term is the surface of idle periods (represented in grey in figure 1).

Depending on the scheduling problem (with or without precedence constraints, unit tasks or not), there are several ways to compute $S_{\text{idle}}$. With precedence constraints, $S_{\text{idle}} \leq (m-1) \cdot T_\infty$ where $T_\infty$ is the length of the longest path of the graph. For independent tasks, the results can be written as $S_{\text{idle}} \leq (m-1) \cdot p_{\max}$ where $p_{\max}$ is the maximum of the processing times. For unit independent tasks, it is straightforward to obtain an optimal algorithm where the global load is evenly balanced. $S_{\text{idle}} = m \cdot \lceil W/m \rceil - W \leq m - 1$, at most one slot of the schedule contains idle times.

When the list of ready tasks is distributed among the processors, the analysis is more complicated even in the elementary case of unit independent tasks. In this case, the extra $S_{\text{idle}}$ term is induced by the distributed nature of the problem. Processors can be idle even when ready tasks are available. Fig. 1 is an example of a schedule obtained using distributed list scheduling which shows the complex repartition of the idle times $S_{\text{idle}}$.

# 3 Model of the Distributed List

Let consider a parallel platform composed of $m$ identical processors and a workload of $n$ tasks with processing times $p_j$. The tasks can be independent or constrained by a directed acyclic graph (DAG) of dependencies. We consider a non-clairvoyant model where the processing times and dependencies are unknown at the beginning of the computation. The problem is to study the maximum completion time (*makespan* denoted by $C_{\max}$) taking into account the scheduling cost.

Instead of storing ready tasks in a centralized queue like in classical list scheduling, each processor $i$ maintains its own local queue $Q_i$ of tasks to execute. When a processor finishes the execution of a task, this task is removed from the queue and another one starts being processed. At the beginning of the execution, ready tasks can be arbitrarily spread among the queues.

We now define the model of execution for managing the distributed list. At every time slot, if the local queue $Q_i$ is not empty, processor $i$ picks a task and executes it. When $Q_i$ is empty, processor $i$ sends a *work request* to a random processor $k$. If $Q_k$ is empty or contains only one task (currently executed by processor $k$), then the request fails and processor $i$ will have to send a new request at the next slot. If $Q_k$ contains more than one task, then $i$ is given half of the tasks and it will restart a normal execution at the next slot. It is important to notice that the underlying mechanisms to send work requests and share work are not specified and could be implemented in many ways.

To model the contention on the queues, which was pointed out before as the bottleneck in centralized list scheduling, no more than one request per queue can succeed in the same time slot. If several requests target the same queue, a random one succeeds and all the others fail.

We now introduce some notations. At time $t$, let $w_i(t)$ be the amount of work in queue $Q_i$, i.e. $w_i(t) = \sum_{j \in Q_i(t)} p_j$, and let $w(t) = \sum_{i=1}^{m} w_i(t)$ be the total work in all queues. The initial workload $W$ is equal to $w(0)$. At time $t$, let $\alpha(t)$ be the number of active processors, i.e. the number of processors with a non-empty queue.

# 4 Principle of the Analysis

The analysis is based on a potential function $\Phi$ representing the load unbalance between the queues. At each time slot, both tasks execution by busy processors ($\Delta_e\Phi$ in lemma 5.1) and work requests by idle processors ($\Delta_r\Phi$ in lemma 5.2) contribute to the diminution of the potential. To compute the potential decrease due to work requests, we compute the expected decrease of the potential due to work requests that targets the same active processor ($\delta_i$ in eq. 1). When the potential reaches 0 no more work requests occurs until the end of the schedule. Using a probabilistic argument, we bound the expected number of work requets $\mathbb{E}[R]$ (theorem 5.3) which leads to an upper bound on the expected makespan $\mathbb{E}[C_{\max}]$ (theorem 5.4) with the geometric analysis presented in section 2.

First, we define a potential function $\Phi(t)$ that is, to a multiplicative factor, the variance of the queue sizes.

**Definition 4.1** *At time t, the potential function $\Phi$ is defined as:*

$$\Phi(t) = \sum_{i=1}^{m} \Big( w_i(t) - \frac{w(t)}{m} \Big)^2$$

*Moreover, let* $\Delta\Phi(t) = \Phi(t) - \Phi(t+1)$.

We report below three useful properties on the potential function $\Phi$.

**Property 4.2** *When* $\Phi(t) = 0$*, we have* $\forall i\ w_i(t) = w(t)/m$*. No more work requests occurs until the end of the schedule as each processor has the same amount of work.*

**Property 4.3** *The potential function is maximal when all the work is in a single queue.*

$$\Phi(t) \leq \Big(1 - \frac{1}{m}\Big) \cdot w(t)^2 \leq \Big(1 - \frac{1}{m}\Big) \cdot W^2$$

**Property 4.4** *The potential function can also be written:*

$$\Phi(t) = \sum_{i=1}^{m} w_i^2(t) - \frac{w^2(t)}{m}$$

# 5 Divisible Workload

In this section, we detail the analysis of DLS on the simplest model, i.e. when the tasks are in a whole divisible workload. It is like independent unit tasks except that the tasks can be divided evenly when a work request occurs, even if the number of tasks is odd (cf. eq. 1).

**Lemma 5.1 (Impact of tasks execution on $\Phi$)** *The execution of one slot of the schedule, without taking into account work requests, decreases the potential function by:*

$$\Delta_e\Phi(t) = \Big(1 - \frac{\alpha(t)}{m}\Big)\Big(2 \cdot w(t) - \alpha(t)\Big) \geq 0$$

This proof is obtained by a direct calculus (see appendix A).

**Lemma 5.2 (Impact of work requests on $\Phi$)** *The expected diminution of the potential function due to work requests can be bounded by*

$$\mathbb{E}\Big[\Delta_r\Phi(t)\Big] \geq \frac{1}{2} \cdot \Big(1 - \Big(1 - \frac{1}{m-1}\Big)^{m-\alpha(t)}\Big) \cdot \Phi(t)$$

**Proof.** Let $\delta_i(t)$ denote the decrease of the potential function due to work requests on processor $i$. If $i$ is not requested then $\delta_i(t) = 0$. If $i$ receives at least one request, we have:

$$\begin{aligned}\delta_i(t) &= \Big(\Big(w_i(t) - \frac{w(t)}{m}\Big)^2 + \Big(0 - \frac{w(t)}{m}\Big)^2\Big) \\ &\quad - \Big(\Big(\frac{w_i(t)}{2} - \frac{w(t)}{m}\Big)^2 + \Big(\frac{w_i(t)}{2} - \frac{w(t)}{m}\Big)^2\Big) \\ &= \frac{w_i(t)^2}{2} \end{aligned} \tag{1}$$

The probability $p_r$ that a processor receives a work request if $m - \alpha(t)$ processors are idle is[1]

$$\begin{aligned} p_r &= 1 - \mathbb{P}\Big\{\bigwedge_{k, w_k(t)=0} i \text{ is not requested by } k\Big\} \\ &= 1 - \Big(1 - \frac{1}{m-1}\Big)^{m-\alpha(t)} \end{aligned} \tag{2}$$

[1] $\mathbb{P}\Big\{\text{event}\Big\}$ denotes the probability of an event

From equations 1 and 2, we can deduce the expected diminution due to work requests on processor $i$.

$$\mathbb{E}\Big[\delta_i(t)\Big] = \frac{w_i(t)^2}{2} \cdot p_r$$

Using the linearity of expectation, we compute the expected diminution of the potential due to all work requests.

$$\mathbb{E}\Big[\Delta_r\Phi(t)\Big] = \mathbb{E}\Big[\sum_{i,w_i(t)\geq 1} \delta_i(t)\Big] = \frac{p_r}{2}\sum_i w_i^2(t) \tag{3}$$

Using property 4.4, we get

$$\begin{aligned}\mathbb{E}\Big[\Delta_r\Phi(t)\Big] &= \frac{1}{2}\cdot p_r \cdot \Big(\Phi(t) + \frac{w^2(t)}{m}\Big) \\ &\geq \frac{1}{2}\cdot p_r \cdot \Phi(t) \qquad \square\end{aligned} \tag{4}$$

**Theorem 5.3 (Work Requests)** *The expected number of work requests $R$ is at most*

$$\mathbb{E}\Big[R\Big] \leq \frac{2e(m-1)}{e-1}\cdot \log_2 \Phi(0) + m - 1$$

**Proof.** We divide the schedule into phases from 1 to $N$ where the potential function is halved in each phase. In phase $i$, we have

$$\frac{\Phi(0)}{2^i} < \Phi(t) \leq \frac{\Phi(0)}{2^{i-1}}$$

and thus, as $\Delta_e\Phi(t) \geq 0$ (cf. lemma 5.1),

$$\begin{aligned}\Delta\Phi(t) &= \Delta_e\Phi(t) + \Delta_r\Phi(t) \geq \Delta_r\Phi(t) \\ &\geq \frac{1}{2}\cdot\Big(1 - \Big(1 - \frac{1}{m-1}\Big)^{m-\alpha(t)}\Big)\cdot\frac{\Phi(0)}{2^i}\end{aligned} \tag{5}$$

Let us denote the function in the right hand side of eq. 5 by $\psi(\alpha)$.

We analyze now the expected number of work requests $R_i$ in phase $i$. We do not take into account time steps where $\alpha(t) = m$ as they do not produce work requests. Notice that $\Delta\Phi(t) = 0$ when $\alpha(t) = m$ (cf. lemma 5.1). Let $t$ denote the time steps when $\alpha(t) \neq m$. Let $T_i$ be the random variable indicating the end of phase $i$,

$$T_i = \min\Big\{\ t \mid \Phi(t) \leq \frac{\Phi(0)}{2^i}\ \Big\}$$

The number of work requests is equal to the number of idle processors at each time step,

$$R_i = \sum_{t=T_{i-1}}^{T_i} m - \alpha(t)$$

As it is difficult to study the sequence of $\alpha(t)$, we will assume that an adversary can choose each $\alpha(t)$ knowing the history of the system but not the future random choices. We construct a Markov decision process which leads to an upper bound on $\mathbb{E}\Big[R_i\Big]$

as follows (see [19] for more details on Markov decision process). The state of the system is $\Phi(t)$. At each epoch, the adversary chooses an action $\alpha(t)$ and is rewarded $m-\alpha(t)$ which corresponds to the number of work requests. The system goes in state $\Phi(t+1) = \Phi(t) - \Delta\Phi(t,\alpha(t))$ where $\Delta\Phi(t,\alpha(t))$ is a random variable corresponding to the diminution of the potential function. Let $u_t(\Phi(t))$ the expected reward obtained following the optimal policy from state $\Phi(t)$ at decision epochs $t, t+1, \ldots, T$. $u_t$ satisfies the optimality equation (Chapter 4 of [19]):

$$u_t(\Phi(t)) = \max_{1\leq\alpha(t)\leq m-1}\Big\{m-\alpha(t)+\sum_{\delta} u_{t+1}(\Phi(t)-\delta)\mathbb{P}\Big\{\Delta\Phi(t,\alpha(t))=\delta\Big\}\Big\}$$

We prove in appendix B using backwards induction on $t$ that

$$u_t(\Phi(t)) \leq \lambda\Big(\Phi(t) - \frac{\Phi(0)}{2^i}\Big)$$

where $\lambda$ is such that

$$\forall\ 1\leq\alpha\leq m-1,\ m-\alpha-\lambda\psi(\alpha)\leq 0 \tag{6}$$

As the optimal policy of the Markov decision process optimizes the expected number of work requests over all sequences of $\alpha(t)$, the expected number of work requests for a fixed sequence of $\alpha(t)$ is bounded by the optimal policy.

$$\begin{aligned}\mathbb{E}\Big[R_i\Big] &\leq u_0\Big(\frac{\Phi(0)}{2^{i-1}}\Big) \leq \lambda\cdot\frac{\Phi(0)}{2^i}\\ &\leq \frac{\Phi(0)}{2^i}\cdot\max_{1\leq\alpha\leq m-1}\frac{m-\alpha}{\frac{1}{2}\cdot p_r\cdot\frac{\Phi(0)}{2^i}}\\ &\leq 2\cdot\frac{e(m-1)}{e-1}\ \text{(cf. appendix C)}\end{aligned} \tag{7}$$

Overall, we can bound the total number of work requests $R'$ until $\Phi(t)\leq 1$.

$$\mathbb{E}\Big[R'\Big] = \sum_{i=1}^{N}\mathbb{E}\Big[R_i\Big] \leq \log_2\Phi(0)\cdot\frac{2e(m-1)}{e-1}$$

In the last phase, i.e. $\Phi(t)\leq 1$, when a processor becomes idle, we have $\forall i, w_i(t)\leq 1$. Thus, there are at most $m-1$ work requests in the last phase, which leads to the claimed bound. □

Finally, we are now able to bound the expected value of the makespan.

**Theorem 5.4** *The expected makespan for a divisible workload $W$ scheduled by DLS is bounded by*

$$\mathbb{E}\Big[C_{\max}\Big] \leq \frac{W}{m} + \frac{4e}{e-1}\cdot\log_2 W + 1$$

**Proof.** As $\Phi(0)\leq(1-\frac{1}{m})\cdot W^2\leq W^2$, we can bound the expected number of work requests by

$$\mathbb{E}\Big[R\Big] \leq \log_2 W^2\cdot\frac{2e(m-1)}{e-1} + m - 1$$

We obtain the claimed bound by using the basic relation $mC_{\max} = W + R$. □

# 6 Independent Unit Tasks

The previous analysis assumes all queues can be evenly divided during a work request. This is not exactly true for queues of odd size. When we take this issue into account, we obtain the following theorem.

**Theorem 6.1** *The expected makespan for $W$ unit independent tasks scheduled by DLS is bounded by*

$$\mathbb{E}\Big[C_{\max}\Big] \leq \frac{W}{m} + \frac{4e}{e-1} \cdot \log_2 W + O(\log m)$$

*This is optimal up to a constant factor in $\log_2 W$.*

**Proof.** In the previous analysis in section 5, eq. 1 can be rewritten for unit non-divisible tasks into

$$\delta_i(t) = w_i(t)^2 - \left\lceil \frac{w_i(t)}{2} \right\rceil^2 - \left\lfloor \frac{w_i(t)}{2} \right\rfloor^2 \geq \frac{w_i(t)^2}{2} - \frac{1}{2}$$

Eq. 4 becomes

$$\mathbb{E}\Big[\Delta_r \Phi(t)\Big] \geq \frac{1}{2} \cdot p_r \cdot \Big(\Phi(t) + \frac{1}{m} w(t)^2 - \alpha(t)\Big)$$

The additional term in $\alpha(t)$ is due to the sum of $\frac{1}{2}$ on all active processors. As $\alpha(t) \leq m$, the lower bound of lemma 5.2 still holds as long as $w(t) \geq m$. When $w(t) < m$, we can obtain a relaxed version of lemma 5.2

$$\mathbb{E}\Big[\Delta_r \Phi(t)\Big] \geq \frac{3}{8} \cdot p_r \cdot \Phi(t) \tag{8}$$

as long as $\alpha(t) \leq w(t)/2$ (see proof in appendix D). As $\alpha(t) > w(t)/2$ can happen only $\log_2 m$ times when $w(t) < m$, we obtain the claimed bound. The overhead in $\mathrm{O}(\log m)$ comes from the relaxed equation 8 applied when $w(t) < m$ and the additional $\log_2 m$ when equation 8 is not valid.

We now give a lower bound for this problem. Consider $W = 2^{k+1}$ tasks and $m = 2^k$ processors. All the tasks are on the same processor at the beginning. In the best case, all work requests target processors with highest loads, the makespan is $C_{\max} = k + 2$. The first $k = \log_2 m$ steps for each processor to get some work, one step where all processors are active and one last step where only one processor is active. Thus $C_{\max} \geq \frac{W}{m} + \log_2 W - 1$. □

# 7 Weighted independent tasks

In this section, we analyze the number of work requests for weighted independent tasks. Each task $j$ has a processing time $p_j$ which is unknown. Let $p_{\min}$ and $p_{\max}$ be the minimum and maximum processing times. During a work request, half of the tasks are transfered from the active processor to the idle processor. As the processing times are unknown, the work cannot be shared evenly between both processors and can be as bad as getting all the smallest tasks. However, we can adapt the previous analysis to the case of weighted tasks and state the following theorem.

**Theorem 7.1** *The expected makespan for weighted independent tasks of total processing time $W$ scheduled by DLS is bounded by*

$$\mathbb{E}\Big[C_{\max}\Big] \leq \frac{W}{m} + O\Big(\frac{p_{\max}}{p_{\min}} \cdot \log W + p_{\max} \cdot \log m\Big)$$

**Proof.** We first examine the load balancing during a work request from processor $j$ to another processor $i$. Assume $i$ has at least two tasks in its queue (at least one task can be transfered). Let $\gamma$ be the fraction of work sent to processor $j$. Eq. 1 becomes

$$\begin{aligned}\delta_i(t) &= \Big(w_i(t) - \frac{w(t)}{m}\Big)^2 + \Big(0 - \frac{w(t)}{m}\Big)^2 \\ &\quad - \Big(\gamma w_i(t) - \frac{w(t}{m}\Big)^2 - \Big((1-\gamma)w_i(t) - \frac{w(t)}{m}\Big)^2 \\ &= 2\gamma(1-\gamma)w_i^2(t)\end{aligned}$$

Processor $j$ receives half of the tasks of processor $i$, but in the worst case we have

$$\gamma = \frac{p_{\min}}{2p_{\max} + p_{\min}}$$

$i$ just started processing one of its two tasks of weight $p_{\max}$ and $j$ gets the last task of weight $p_{\min}$. When $i$ has only one task currently in process (no task can be transfered) the potential function does not decrease. As this task can be of weight $p_{\max}$, we have

$$\begin{aligned}\delta_i(t) &\geq 2\gamma(1-\gamma)\cdot(w_i^2(t) - p_{\max}^2) \\ &\geq 4\cdot\frac{p_{\min}p_{\max}}{(2p_{\max}+p_{\min})^2}\cdot(w_i^2(t) - p_{\max}^2) \\ &\geq \frac{4}{9}\cdot\frac{p_{\min}}{p_{\max}}\cdot(w_i^2(t) - p_{\max}^2)\end{aligned}$$

We obtain a new version of lemma 5.2,

$$\begin{aligned}\mathbb{E}\Big[\Delta_r\Phi(t)\Big] &\geq \frac{4}{9}\cdot\frac{p_{\min}}{p_{\max}}\cdot p_r\cdot\sum_{i,w_i(t)\geq 1}\Big(w_i^2(t) - p_{\max}^2\Big) \\ &\geq \frac{4}{9}\cdot\frac{p_{\min}}{p_{\max}}\cdot p_r\cdot\Big(\Phi(t) + \frac{w^2(t)}{m} - \alpha p_{\max}^2\Big)\end{aligned}$$

As long as $w(t) \geq mp_{\max}$,

$$\mathbb{E}\Big[\Delta_r\Phi(t)\Big] \geq \frac{4}{9}\cdot\frac{p_{\min}}{p_{\max}}\cdot p_r\cdot\Phi(t)$$

and when $w(t) < mp_{\max}$, we can obtain a relaxed version of the bound like in section 6. The overhead due to non-divisible tasks becomes $\mathrm{O}(p_{\max}\log m)$ and we obtain the expected makespan.

$$\mathbb{E}\Big[C_{\max}\Big] \quad \leq \quad \frac{W}{m} \;+\; \frac{9e}{2(e-1)} \;\cdot\; \frac{p_{\max}}{p_{\min}} \;\cdot\; \log_2\Phi_0 \;+\; \mathrm{O}(p_{\max} \;\cdot\; \log m)$$

As in theorem 5.4, we can bound $\Phi_0$ by $W^2$, which leads to the result. □

# 8 Tasks with Precedences

In this section, we show how the well known non-blocking work stealing of Arora *et al.* [3] (denoted ABP in the sequel) can be analyzed with our method which gives a slightly tighter bound for the expected makespan. Following [3], a multithreaded

computation is modeled as a directed acyclic graph $G$ with a single root node; each node corresponds to a unit task and edges define precedence constraints. The out-degree of each node is either 0, 1 or 2. The critical path of $G$ is denoted $T_\infty$ and $W$ is its total number of nodes.

ABP schedules the DAG $G$ as follows. Each process $i$ maintains a double-ended queue – deque – $Q_i$ of ready nodes (the notion of process here corresponds to our processors). At each top, an active process $i$ with a non-empty deque executes the node at the bottom of its deque $Q_i$; once its execution is completed, this node is popped from the bottom of the deque, enabling – i.e. making ready – 0, 1 or 2 child nodes that are pushed at the bottom of $Q_i$. At each top, an idle process $j$ with an empty deque $Q_j$ becomes a theft: it performs a work request on another randomly chosen victim deque; if the victim deque contains ready nodes, then its top-most node is popped and pushed into the deque of one of its concurrent thefts. If $j$ becomes active just after its work request, the work request is said successful. Otherwise, $Q_j$ remains empty and the work request failed which may occur in the three following situations: either the victim deque $Q_i$ is empty; or, $Q_i$ contains only one node currently in execution on $i$; or, due to contention, another theft performs a successful work request on $i$ simultaneously.

Let us first recall the definition of the *enabling tree* of [3]. If the execution of node $u$ enables node $v$, then the edge $(u, v)$ of $G$ is an enabling edge. The sub-graph of $G$ consisting of only enabling edges forms a rooted tree called the enabling tree. Following [3], if $d(u)$ is the depth of a node $u$ in the enabling tree, then its weight is defined as $\omega(u) = T_\infty - d(u)$. The weight of the root node is $T_\infty$.

To represent the amount of work on each processor, we use a *potential work* $w_i(t) = 2^{\max\{w(u):u\in Q_i(t)\}}$, i.e. the maximum number of nodes that can be activated by a task in $Q_i$. We first need to study the repartition of the potential work during a work request (the equivalent of eq. 1).

**Lemma 8.1** *For any active process $i$, we have $w_i(t+1) \leq w_i(t)$. Moreover, after any work request from a process $j$ on $i$,*

$$w_i(t+1) \leq \frac{w_i(t)}{2} \text{ and } w_j(t+1) \leq \frac{w_i(t)}{2}.$$

**Proof.** The proof is derived from [3], corollary 4 in section 3: if at $t$, $Q_i$ contains the $k+1$ nodes $v_0, v_1, \ldots, v_k$ from bottom to top, then $\omega(v_0) \leq \omega(v_1) < \ldots < \omega(v_{k-1}) < \omega(v_k)$. After the execution of a node $u$, the maximum weight of its two enabled children is less than $\omega(u) - 1$. Thus the potential work $w_i$ cannot increase.

We now state that the potential is halved after any work request by distinguishing two cases. First, when a successful steal occurs on $i$ from $j$, then the node $v_k$ has been stolen and executed by $j$. Thus, either $w_i(t+1) = 0$ if $Q_i$ is empty at $t+1$; or $w_i(t+1) = 2^{\omega(v_{k-1})} \leq 2^{\omega(v_k)-1} \leq w_i(t)/2$. Besides, after execution of $v_k$ by $j$, $w_j(t+1) \leq 2^{\omega(v_k)-1} \leq w_i(t)/2$. Secondly, if all work requests that occur on $i$ are unsuccessful, then there was only one node $v_0$ in $Q_i$ whose execution was processed by $i$. Then, at $t+1$, $w_i(t+1) \leq 2^{\omega(v_0)-1} \leq w_i(t)/2$ and $w_j(t+1) = 0$. □

We can now state the following theorem.

**Theorem 8.2** *The expected makespan verifies:*

$$\mathbb{E}\Big[C_{\max}\Big] \leq \frac{W}{m} + \frac{4e}{e-1}T_\infty + 1 < \frac{W}{m} + 6.33T_\infty + 1.$$

**Proof.** The proof is a direct application of theorems 5.3 and 5.4 using a modified potential function. Note that we cannot use directly the same as before (defined in 4.1) because the total amount of potential work may be reduced when a steal occurs[2]. Yet let $\Phi(t) = \sum_i w_i^2(t)$ be the potential. From lemma 8.1, we know that the reduction of the potential due to a work request on $i$ is at least $\delta_i(t) \geq w_i^2(t)/2$, thus eq. 3 becomes $\mathbb{E}\Big[\Delta_r\Phi(t)\Big] \geq p_r\Phi(t)/2$. As $\Phi(0) = (2^{2T_\infty})$, from theorem 5.3, we get the expected number of work requests $\mathbb{E}\Big[R\Big] \leq \frac{4e(m-1)}{e-1} \cdot T_\infty + m - 1$ and the expected makespan $\mathbb{E}\Big[C_{\max}\Big] \leq \frac{W}{m} + \frac{4e}{e-1} \cdot T_\infty + 1$ which completes the proof. □

**Remark.** Additionally, [3] stated that $C_{\max} - \frac{W}{m} = \mathrm{O}(T_\infty)$ with high probability, but also established the upper bound $\mathbb{E}\Big[C_{\max} - \frac{W}{m}\Big] < 32T_\infty$ (see [3], proof of theorem 9 in section 4.3). Yet, the upper bound $\mathbb{E}\Big[C_{\max} - \frac{W}{m}\Big] < 6.33T_\infty + 1$ improves the previous upper bound from [3].

# 9 Initial Repartition of Tasks

In the previous sections, we assumed that all tasks are on the same processor at the beginning of the execution. In this section, we use the bounds in terms of $\Phi_0$ to show that a more balanced initial repartition leads to fewer work requests on average.

Let us first focus on unit independent tasks. Recall the result of theorem 6.1 for the worst initial repartition. We take a balls-and-bins assignment as the initial repartition. For each task, we choose uniformly a random processor and put the task in its queue. We compute the expected value of $\Phi_0$ using property 4.4.

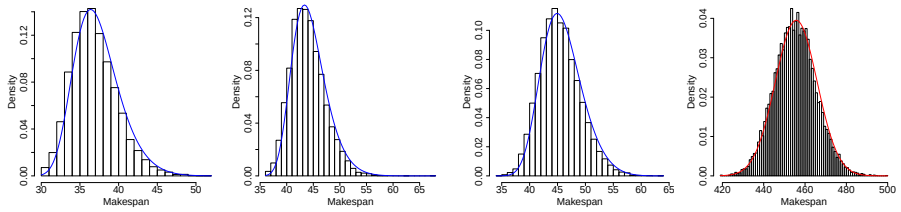
$$\begin{aligned}
\mathbb{E}\Big[\Phi_0\Big] &= \sum_i \mathbb{E}\Big[w_i^2\Big] - \frac{W^2}{m} \\
&= \sum_i \Big(\mathrm{Var}\Big[w_i\Big] + \mathbb{E}\Big[w_i\Big]^2\Big) - \frac{W^2}{m} \\
&= m\mathrm{Var}\Big[w_1\Big] = \Big(1 - \frac{1}{m}\Big) \cdot W
\end{aligned}$$

as $w_i$ follows a binomial distribution. Since the number of work requests is proportional to $\log_2 \Phi_0$, this initial distribution of tasks reduces the number of work requests by a factor of 2 on average.

Let now consider weighted tasks. One can show that putting task $j$ on a random processor increases $\Phi_0$ by $(1 - \frac{1}{m}) \cdot p_j^2$ on average and thus, we obtain

$$\begin{aligned}
\mathbb{E}\Big[\Phi_0\Big] &= \Big(1 - \frac{1}{m}\Big) \cdot \sum_j p_j^2 \\
&\leq \Big(1 - \frac{1}{m}\Big) \cdot \sum_j p_j p_{\max} \\
&\leq \Big(1 - \frac{1}{m}\Big) \cdot W \cdot p_{\max}
\end{aligned}$$

[2]This can happen when the sibling of the stolen task has only one child.




(a) Unit Tasks (b) Weighted Tasks (c) DAG (short $T_\infty$) (d) DAG (long $T_\infty$)

Figure 2: Distribution of the makespan for unit independent tasks 2(a), weighted independent tasks 2(b) and tasks with dependencies 2(c) and 2(d). The first three models follow a gev distribution (blue curves), the last one is gaussian (red curve).

which reduces the number of work requests of theorem 7.1 by roughly a factor of 2 when $p_{\max}$ is much lower than $W$.

The case with precedence constraints is not analyzed here because there is only one task initially (the unique source of the DAG).

# 10 Experimental Study

The theoretical analysis gives an upper bound on the expected value of the makespan for the various models we considered. In this section, we study experimentally the distribution of the makespan. Statistical tests give evidence that the makespan for independent tasks follows a generalized extreme value (gev) distribution [16]. This was expected since such a distribution arises when dealing with maximum of random variables. For tasks with dependencies, it depends on the structure of the graph: DAGs with short critical path still follow a gev distribution but when the critical path grows, it tends to a gaussian distribution. We also study in more details the overhead to $W/m$ and show that it is approximately $3\log_2 W$ for unit independent tasks which is close to the theoretical result of $\frac{4e}{e-1}\log_2 W$.

We developed a simulator that strictly follows our model. At the beginning, all the tasks are given to processor 0 in order to be in the worst case, i.e. when the initial potential $\Phi_0$ is maximum.

**Distribution of the Makespan.** We consider here a fixed workload $W = 2^{17}$ on $m = 2^{10}$ processors for independent tasks and $m = 2^7$ processors for tasks with dependencies. For the weighted model, processing times were generated randomly and uniformly between 1 and 10. For the DAG model, graphs have been generated using a layer by layer method. We generated two types of DAGs, one with a short critical path (close to the minimum possible $\log_2 W$) and the other one with a long critical path (around $W/4m$ in order to keep enough tasks per processor per layer). For each workload (unit, weighted and DAG), we executed the simulator $10,000$ times to obtain the distribution of the makespan. Fig. 2 presents histograms for $C_{\max} - \lceil W/m \rceil$.

The distributions of the first three models (a,b,c in fig. 2) are clearly not gaussian: they are asymmetrical with an heavier right tail. To fit these three models, we use the generalized extreme value (gev) distribution [16]. In the same way as the normal distribution arises when studying the sum of independent and identically distributed (iid)

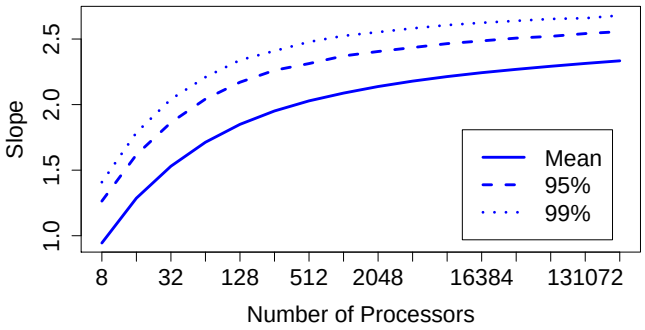



Figure 3: Proportionality constant of $\log_2 W$ against the number of processors (in logarithmic scale).

random variables, the gev distribution arises when studying the maximum of iid random variables. The extreme value theorem, an equivalent of the central limit theorem for maxima, states that the maximum of iid random variables converges in distribution to a gev distribution. In our setting, the random variables measuring the load of each processor are not independent, thus the extreme value theorem cannot apply directly. However, it is possible to fit the distribution of the makespan to a gev distribution. In fig. 2, the fitted distributions (blue curve) closely follow the histograms. To confirm this graphical approach, we performed a goodness of fit test. The $\chi^2$ test is well-suited to our data because the distribution of the makespan is discrete. We compared the results of the best fitted gev to the best fitted gaussian. The $\chi^2$ test strongly rejects the gaussian hypothesis but does not reject the gev hypothesis with a p-value of more than $0.5$. This confirms that the makespan follows a gev distribution. We fitted the last model, DAG with long critical path, with a gaussian (red curve in fig. 2(d)). In this last case, the completion time of each layer of the DAG should correspond to a gev distribution but the total makespan, the sums of all layers, should tend to a gaussian by the central limit theorem. Indeed the $\chi^2$ test does not reject the gaussian hypothesis with a p-value around $0.3$.

**Study of the $\log_2 W$ term.** We focus now on unit independent tasks as the other models rely on too many parameters (the choice of the processing times for weighted tasks and the structure of the DAG for tasks with dependencies). We want to show that the number of work requests is proportional to $\log_2 W$ and study the proportionality constant. We first launch simulations with a fixed number of processors $m$ and a wide range of work in successive powers of two. A linear regression confirms the linear dependency in $\log_2 W$ with a coefficient of determination ("r squared") greater than $0.9999$[3].

Then, we obtain the slope of the regression for various number of processors. The value of the slope tends to a limit between 2 and 3 (cf. solid curve in fig. 3). This shows that the theoretical analysis of theorem 6.1 is almost accurate with a constant of $\frac{4e}{e-1} \approx 6.33$. The difference between the theoretical and the practical value can be explained by the use of an adversary in theorem 5.3.

We also compute the slope taking the quantile at 99% instead of the mean (i.e. 99% of the $10,000$ simulations have a smaller makespan than the considered value).

[3] the closest to 1, the better

This experiment shows that the constant factor of $\log_2 W$ is less than 3 in 99% of all simulations (cf. dotted curve in fig. 3).

# 11 Concluding Remarks

We have presented in this paper a new analysis of the distributed list scheduling algorithms. It was presented in detail for the case of a global divisible workload $W$ on $m$ processors. The main result was to prove that the expected makespan is no more than the best possible absolute lower bound $W/m$ plus an additive term in $\frac{4e}{e-1}\log_2 W$ very close to the value obtained by simulation. This technique was successfully extended for unit and weighted independent tasks and for tasks with dependencies. In all cases, we succeeded to characterize precisely the overhead due to the decentralization of the list of ready tasks. We also believe that this work will help to clarify the links between classical list scheduling and work stealing. Another interesting issue is that we developed a simulator for better characterizing the behaviour of the DLS algorithms. In particular, we gave statistical evidence that the distribution of the makespan for independent tasks follows a gev distribution.

Based on the results of this paper, it remains challenging open problems to study. First, in the case of weighted independent tasks, it would be interesting to study the impact of local policies to manage the queues on the global performance. It would also be of interest to analyze a modified work stealing algorithm where idle processors steal tasks using an affinity criterion. We believe that our analysis is well-suited for affinity scheduling because processors are free to steal any tasks of the queue.

# References

[1] U. A. Acar, G. E. Blelloch, and R. D. Blumofe. The data locality of work stealing. In *Proceedings of SPAA'00*, pages 1–12, 2000.

[2] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. In *Proceedings of STOC '95*, pages 238–247, 1995.

[3] N. S. Arora, R. D. Blumofe, and C. G. Plaxton. Thread scheduling for multiprogrammed multiprocessors. *Theory of Computing Systems*, 34(2):115–144, 2001.

[4] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[5] P. Berenbrink, T. Friedetzky, and L. A. Goldberg. The natural work-stealing algorithm is stable. *SIAM Journal of Computing*, 32(5):1260–1279, 2003.

[6] P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. W. Goldberg, Z. Hu, and R. Martin. Distributed selfish load balancing. *SIAM Journal on Computing*, 37(4):1163–1181, 2007.

[7] P. Berenbrink, T. Friedetzky, and Z. Hu. A new analytical method for parallel, diffusion-type load balancing. *Journal of Parallel and Distributed Computing*, 69(1):54 – 61, 2009.

[8] P. Berenbrink, T. Friedetzky, Z. Hu, and R. Martin. On weighted balls-into-bins games. *Theoretical Computer Science*, 409(3):511 – 520, 2008.

[9] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

[10] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41(2):212 – 224, 2001.

[11] M. Drozdowski. *Scheduling for Parallel Processing*. Springer, 2009.

[12] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *Proceedings of PLDI'98*, 1998.

[13] T. Gautier, X. Besseron, and L. Pigeon. KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *Proceedings of PASCO'07*, pages 15–23, 2007.

[14] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[15] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, 18(2):244–257, 1989.

[16] S. Kotz and S. Nadarajah. *Extreme Value Distributions: Theory and Applications*. World Scientific Publishing Company, 2001.

[17] J. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[18] M. Mitzenmacher. Analyses of load stealing models based on differential equations. In *Proceedings of SPAA'98*, pages 212–221, 1998.

[19] M. L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[20] Y. Robert and F. Vivien. *Introduction to Scheduling*. Chapman & Hall/CRC Press, 2009.

[21] A. Robison, M. Voss, and A. Kukanov. Optimization via reflection on work stealing in TBB. In *Proceedings of IPDPS'08*, pages 1–8, 2008.

[22] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour. Online scheduling in grids. In *Proceedings of IPDPS'08*, April 2008.

[23] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Proceedings of the Eighth Heterogeneous Computing Workshop*, page 3, 1999.

[24] D. Traoré, J.-L. Roch, N. Maillard, T. Gautier, and J. Bernard. Deque-free work-optimal parallel stl algorithms. In *Proceedings of Euro-Par'08*, pages 887–897, 2008.

# A Proof of lemma 5.1

Each of the $\alpha(t)$ active processors execute one task, $w(t+1) = w(t) - \alpha(t)$.

$$\begin{aligned}
\Phi(t+1) &= \sum_{i,q_i(t)\geq 1} \Big(q_i(t) - 1 - \frac{w(t)-\alpha(t)}{m}\Big)^2 + (m-\alpha(t))\Big(0 - \frac{w(t)-\alpha(t)}{m}\Big)^2 \\
&= \sum_{i,q_i(t)\geq 1} \Big(q_i(t) - \frac{w(t)}{m} - \Big(1 - \frac{\alpha(t)}{m}\Big)\Big)^2 \\
&\quad + (m-\alpha(t))\Big(0 - \frac{w(t)}{m} + \frac{\alpha(t)}{m}\Big)^2 \\
&= \Phi(t) - 2\cdot\Big(1-\frac{\alpha(t)}{m}\Big)\cdot \sum_{i,q_i(t)\geq 1}\Big(q_i(t) - \frac{w(t)}{m}\Big) + \alpha(t)\Big(1-\frac{\alpha(t)}{m}\Big)^2 \\
&\quad + (m-\alpha(t))\Big(\Big(\frac{\alpha(t)}{m}\Big)^2 - 2\cdot\frac{\alpha(t)}{m}\cdot\frac{w(t)}{m}\Big)
\end{aligned}$$

Thus,

$$\begin{aligned}
\Delta_e\Phi(t) &= 2\cdot\Big(1-\frac{\alpha(t)}{m}\Big)\cdot \sum_{i,q_i(t)\geq 1}\Big(q_i(t) - \frac{w(t)}{m}\Big) - \alpha(t)\Big(1-\frac{\alpha(t)}{m}\Big)^2 \\
&\quad - (m-\alpha(t))\Big(\Big(\frac{\alpha(t)}{m}\Big)^2 + 2\cdot\frac{\alpha(t)}{m}\cdot\frac{w(t)}{m}\Big) \\
&= \Big(1-\frac{\alpha(t)}{m}\Big)\Big(2\sum_{i,q_i(t)\geq 1}\Big(q_i(t) - \frac{w(t)}{m}\Big) \\
&\quad - \alpha(t)\cdot\Big(1-\frac{\alpha(t)}{m}\Big) - \frac{\alpha(t)^2}{m} + 2\cdot\frac{\alpha(t)}{m}\cdot w(t)\Big) \\
&= \Big(1-\frac{\alpha(t)}{m}\Big)\Big(2\cdot w(t) - 2\cdot\alpha(t)\cdot\frac{w(t)}{m} \\
&\quad - \alpha(t) + \frac{\alpha(t)^2}{m} - \frac{\alpha(t)^2}{m} + 2\cdot\frac{\alpha(t)}{m}\cdot w(t)\Big) \\
&= \Big(1-\frac{\alpha(t)}{m}\Big)\Big(2\cdot w(t) - \alpha(t)\Big)
\end{aligned}$$

Moreover, as $\alpha(t) \leq m$ and $w(t) \geq \alpha(t)$, we have $\Delta_e\Phi(t) \geq 0$.

# B Induction step of theorem 5.3

We prove using backwards induction on $t$ that

$$u_t(\Phi(t)) \leq \lambda\Big(\Phi(t) - \frac{\Phi(0)}{2^i}\Big)$$

where $\lambda$ is such that

$$\forall\ 1 \leq \alpha \leq m-1,\ m - \alpha - \lambda\psi(\alpha) \leq 0$$

At the end of phase $i$, $u_T(\frac{\Phi(0)}{2^i}) = 0$ as there are no more work requests in this phase. $u_T$ satisfies the base case. We prove the induction step in the following.

$$\begin{aligned}
u_t(\Phi(t)) &= \max_{1\le\alpha(t)\le m-1}\Big\{m-\alpha(t)+\sum_{\delta} u_{t+1}(\Phi(t)-\delta)\mathbb{P}\Big\{\Delta\Phi(t,\alpha(t))=\delta\Big\}\Big\}\\
&\le \max_{1\le\alpha(t)\le m-1}\Big\{m-\alpha(t)+\sum_{\delta}\lambda\Big(\Phi(t)-\delta-\frac{\Phi(0)}{2^i}\Big)\mathbb{P}\Big\{\Delta\Phi(t,\alpha(t))=\delta\Big\}\Big\}\\
&= \max_{1\le\alpha(t)\le m-1}\Big\{m-\alpha(t)+\lambda\Big(\Phi(t)-\frac{\Phi(0)}{2^i}\Big)\sum_{\delta}\mathbb{P}\Big\{\Delta\Phi(t,\alpha(t))=\delta\Big\}\\
&\qquad\qquad -\lambda\sum_{\delta}\delta\mathbb{P}\Big\{\Delta\Phi(t,\alpha(t))=\delta\Big\}\Big\}\\
&= \max_{1\le\alpha(t)\le m-1}\Big\{m-\alpha(t)+\lambda\Big(\Phi(t)-\frac{\Phi(0)}{2^i}\Big)-\lambda\mathbb{E}\Big[\Delta\Phi(t,\alpha(t))\Big]\Big\}\\
&= \max_{1\le\alpha(t)\le m-1}\Big\{m-\alpha(t)+\lambda\Big(\Phi(t)-\frac{\Phi(0)}{2^i}\Big)-\lambda\psi(\alpha(t))\Big\}\\
&= \lambda\Big(\Phi(t)-\frac{\Phi(0)}{2^i}\Big)+\max_{1\le\alpha(t)\le m-1}\Big\{m-\alpha(t)-\lambda\psi(\alpha(t))\Big\}\\
&\le \lambda\Big(\Phi(t)-\frac{\Phi(0)}{2^i}\Big) \qquad \text{(by definition of } \lambda \text{ in eq. 6)}
\end{aligned}$$

# C Proof of inequality 7 of theorem 5.3

Let define $f$ by

$$f(\alpha)=\frac{m-\alpha}{1-\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}}$$

and compute the derivative $f'$.

$$\begin{aligned}
&\Big(1-\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}\Big)^2\cdot f'(\alpha)\\
&\quad=-\Big(1-\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}\Big)\\
&\qquad+(m-\alpha)\cdot\ln\Big(1-\frac{1}{m-1}\Big)\cdot\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}\\
&\quad=-1+\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}\cdot\Big(1+(m-\alpha)\ln\Big(1-\frac{1}{m-1}\Big)\Big)\\
&\quad\le-1+\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}\cdot\Big(1+(m-\alpha)\frac{-1}{m-1}\Big)\\
&\quad\le-1+\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}\cdot\frac{\alpha-1}{m-1}\le 0
\end{aligned}$$

As $f$ is non increasing for $1\le\alpha\le m-1$,

$$\max_{1\le\alpha\le m-1}\frac{m-\alpha}{1-\Big(1-\frac{1}{m-1}\Big)^{m-\alpha}}=f(1)=\frac{m-1}{1-\Big(1-\frac{1}{m-1}\Big)^{m-1}}$$

Moreover

$$\Big(1-\frac{1}{m-1}\Big)^{m-1}=\exp\Big((m-1)\ln\Big(1-\frac{1}{m-1}\Big)\Big)\le\exp\Big((m-1)\cdot\frac{-1}{m-1}\Big)\Big)\le\frac{1}{e}$$

and thus

$$\max_{1\leq\alpha\leq m-1}\frac{m-\alpha}{1-\left(1-\frac{1}{m-1}\right)^{m-\alpha}}\leq\frac{m-1}{1-\frac{1}{e}}\leq\frac{e(m-1)}{e-1}$$

# D Proof of eq. 8

When $m-\alpha(t)$ processors have an empty queue, we can bound $\Phi(t)$ by

$$\begin{aligned}\Phi(t)&\geq(m-\alpha(t))\cdot\left(0-\frac{w(t)}{m}\right)^2+\alpha(t)\cdot\left(\frac{w(t)}{\alpha(t)}-\frac{w(t)}{m}\right)^2\\&\geq w^2\cdot\left(\frac{1}{\alpha(t)}-\frac{1}{m}\right)\end{aligned}\tag{9}$$

We study in the following the sign of expression $A$

$$A=\alpha(t)-\frac{w(t)^2}{m}-\frac{w(t)^2}{4}\cdot\left(\frac{1}{\alpha(t)}-\frac{1}{m}\right)$$

which as the same sign as

$$\alpha(t)^2-\frac{w(t)^2}{m}\cdot\left(1-\frac{1}{4m}\right)\cdot\alpha-\frac{w(t)^2}{4}$$

We compute the $\Delta$ of this polynomial in $\alpha(t)$

$$\Delta=\frac{w(t)^4}{m^2}\left(1-\frac{1}{4m}\right)^2+w(t)^2>0$$

We denote the negative root by $\alpha_1$ and the positive root by $\alpha_2$.

$$\alpha_2=\frac{1}{2}\cdot\left(\frac{w(t)^2}{m}\cdot\left(1-\frac{1}{4m}\right)+\sqrt{\Delta}\right)>\frac{w(t)}{2}\qquad(\text{as }\Delta>w(t)^2)$$

So expression $A$ is negative when $\alpha(t)\leq\alpha_2$ and positive when $\alpha(t)>\alpha_2$. Thus when $\alpha(t)\leq w(t)/2$,

$$\begin{aligned}\alpha(t)-\frac{w(t)^2}{m}-\frac{w(t)^2}{4}\cdot\left(\frac{1}{\alpha(t)}-\frac{1}{m}\right)&\leq 0\\\alpha(t)-\frac{w(t)^2}{m}-\frac{1}{4}\cdot\Phi(t)&\leq 0\qquad(\text{c.f. eq. 9})\\\Phi(t)+\frac{w(t)^2}{m}-\alpha(t)&\geq\frac{3}{4}\cdot\Phi(t)\\\mathbb{E}\Big[\Delta_r\Phi(t)\Big]&\geq\frac{3}{8}\cdot\left(1-\left(1-\frac{1}{m-1}\right)^{m-\alpha(t)}\right)\cdot\Phi(t)\end{aligned}$$

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399